Application.prperties

server.port=9292

spring.datasource.driver-class-name=com.mysql.cj.jdbc.Driver

spring.jpa.properties.hibernate.dialect=org.hibernate.dialect.MySQL8Dialect

spring.jpa.hibernate.ddl-auto=update

spring.datasource.url=jdbc:mysql://localhost:3306/blog

spring.datasource.username=root

spring.datasource.password=root

spring.jpa.show-sql=true

**Create entities package**

package com.example.springbackend.entities;

import javax.persistence.Column;

import javax.persistence.Entity;

import javax.persistence.GeneratedValue;

import javax.persistence.GenerationType;

import javax.persistence.Id;

import javax.persistence.Table;

@Entity

@Table(name = "users")

public class User {

@Id

@GeneratedValue(strategy = GenerationType.AUTO)

private int id;

@Column(unique = true)

private String email;

private String name;

private String password;

private String about;

//setters and getters

}

**Create repo package**

package com.example.springbackend.repo;

import org.springframework.data.jpa.repository.JpaRepository;

import org.springframework.stereotype.Repository;

import com.example.springbackend.entities.\*;

@Repository

public interface UserRepo extends JpaRepository<User,Integer> {

}

public class UserDto {

private int id;

private String email;

private String name;

private String password;

private String about;

//setters and getter

}

**Create service package**

package com.example.springbackend.services;

import java.util.List;

import com.example.springbackend.dto.UserDto;

public interface UserService {

UserDto createUser(UserDto user);

UserDto updateUser(UserDto user, Integer userId);

UserDto getUserById(Integer userId);

List<UserDto> getAllUsers();

void deleteUser(Integer userId);

}

**To handle the exceptions**

package com.example.springbackend.exceptions;

public class ResourceNotFoundException extends RuntimeException {

String resourceName;

String fieldName;

long fieldValue;

public ResourceNotFoundException(String resourceName, String fieldName, long fieldValue) {

super(String.format("%s not found with %s : %s", resourceName, fieldName, fieldValue));

this.resourceName = resourceName;

this.fieldName = fieldName;

this.fieldValue = fieldValue;

}

//setters and getters

}

public class ApiResponse {

private String message;

private boolean success;

//getters and setters

}

public class ApiException extends RuntimeException {

public ApiException(String message) {

super(message);

}

public ApiException() {

super();

}

}

package com.example.springbackend.services.impl;

import java.util.List;

import java.util.stream.Collectors;

import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.stereotype.Service;

import com.example.springbackend.dto.UserDto;

import com.example.springbackend.entities.User;

import com.example.springbackend.exceptions.ResourceNotFoundException;

import com.example.springbackend.repo.UserRepo;

import com.example.springbackend.services.UserService;

@Service

public class UserServiceImpl implements UserService {

@Autowired

private UserRepo userRepo;

@Override

public UserDto createUser(UserDto userDto) {

User user = this.dtoToUser(userDto);

User savedUser = this.userRepo.save(user);

return this.userToDto(savedUser);

}

@Override

public UserDto updateUser(UserDto userDto, Integer userId) {

User user = this.userRepo.findById(userId)

.orElseThrow(() -> new ResourceNotFoundException("User", " Id ", userId));

user.setName(userDto.getName());

user.setEmail(userDto.getEmail());

user.setPassword(userDto.getPassword());

user.setAbout(userDto.getAbout());

User updatedUser = this.userRepo.save(user);

UserDto userDto1 = this.userToDto(updatedUser);

return userDto1;

}

@Override

public UserDto getUserById(Integer userId) {

User user = this.userRepo.findById(userId)

.orElseThrow(() -> new ResourceNotFoundException("User", " Id ", userId));

return this.userToDto(user);

}

@Override

public List<UserDto> getAllUsers() {

List<User> users = this.userRepo.findAll();

List<UserDto> userDtos = users.stream().map(user -> this.userToDto(user)).collect(Collectors.toList());

return userDtos;

}

@Override

public void deleteUser(Integer userId) {

User user = this.userRepo.findById(userId)

.orElseThrow(() -> new ResourceNotFoundException("User", "Id", userId));

this.userRepo.delete(user);

}

public User dtoToUser(UserDto userDto) {

User user = new User();

user.setId(userDto.getId());

user.setName(userDto.getName());

user.setEmail(userDto.getEmail());

user.setAbout(userDto.getAbout());

user.setPassword(userDto.getPassword());

return user;

}

public UserDto userToDto(User user) {

UserDto userDto = new UserDto();

userDto.setId(user.getId());

userDto.setName(user.getName());

userDto.setEmail(user.getEmail());

userDto.setAbout(user.getAbout());

userDto.setPassword(user.getPassword());

return userDto;

}

}

package com.example.springbackend.controller;

import java.util.List;

import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.http.HttpStatus;

import org.springframework.http.ResponseEntity;

import org.springframework.web.bind.annotation.DeleteMapping;

import org.springframework.web.bind.annotation.GetMapping;

import org.springframework.web.bind.annotation.PathVariable;

import org.springframework.web.bind.annotation.PostMapping;

import org.springframework.web.bind.annotation.PutMapping;

import org.springframework.web.bind.annotation.RequestBody;

import org.springframework.web.bind.annotation.RequestMapping;

import org.springframework.web.bind.annotation.RestController;

import com.example.springbackend.dto.UserDto;

import com.example.springbackend.exceptions.ApiResponse;

import com.example.springbackend.services.UserService;

@RestController

@RequestMapping("/api/users")

public class UserController {

@Autowired

private UserService userService;

// POST-create user

@PostMapping("/save")

public ResponseEntity<UserDto> createUser( @RequestBody UserDto userDto) {

UserDto createUserDto = this.userService.createUser(userDto);

return new ResponseEntity<>(createUserDto, HttpStatus.CREATED);

}

// PUT- update user

@PutMapping("/update/{userId}")

public ResponseEntity<UserDto> updateUser(@RequestBody UserDto userDto, @PathVariable("userId") Integer uid) {

UserDto updatedUser = this.userService.updateUser(userDto, uid);

return ResponseEntity.ok(updatedUser);

}

@DeleteMapping("/delete/{userId}")

public ResponseEntity<ApiResponse> deleteUser(@PathVariable("userId") Integer uid) {

this.userService.deleteUser(uid);

return new ResponseEntity<ApiResponse>(new ApiResponse("User deleted Successfully", true), HttpStatus.OK);

}

// GET - user get

@GetMapping("/getalldata")

public ResponseEntity<List<UserDto>> getAllUsers() {

return ResponseEntity.ok(this.userService.getAllUsers());

}

// GET - user get

@GetMapping("/getuser/{userId}")

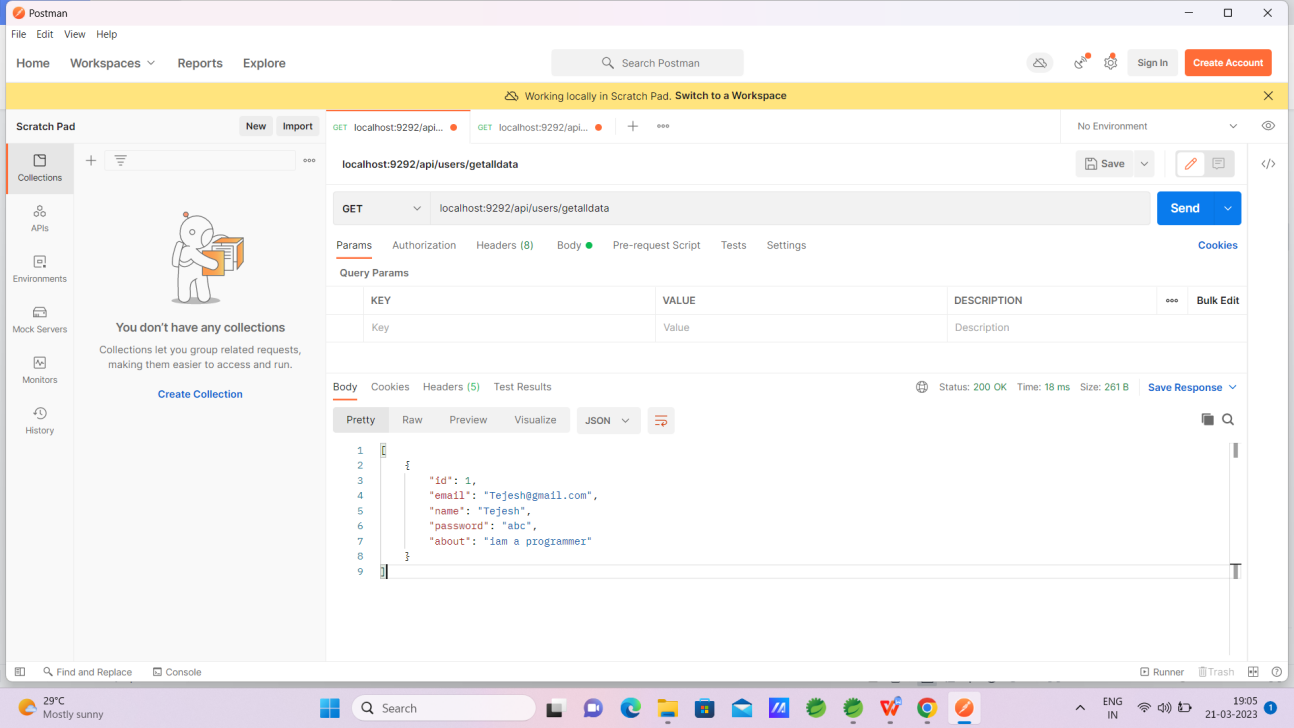
public ResponseEntity<UserDto> getSingleUser(@PathVariable Integer userId) {

return ResponseEntity.ok(this.userService.getUserById(userId));

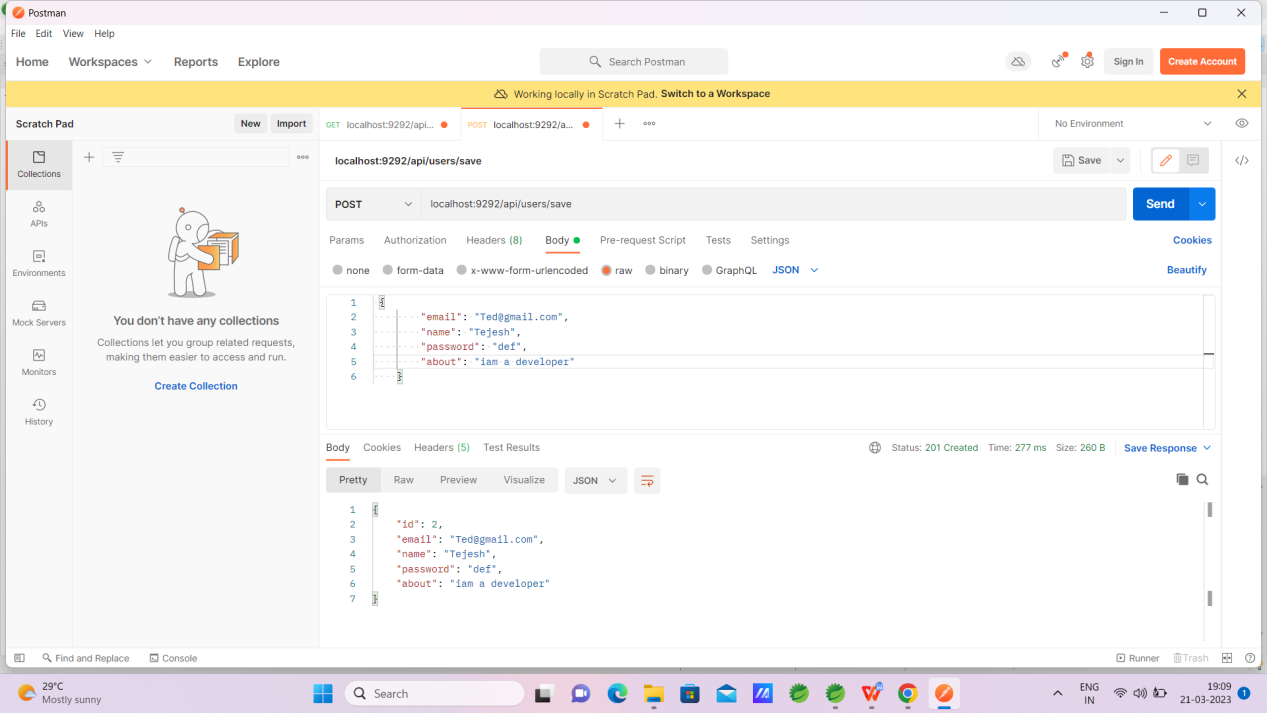
}

}

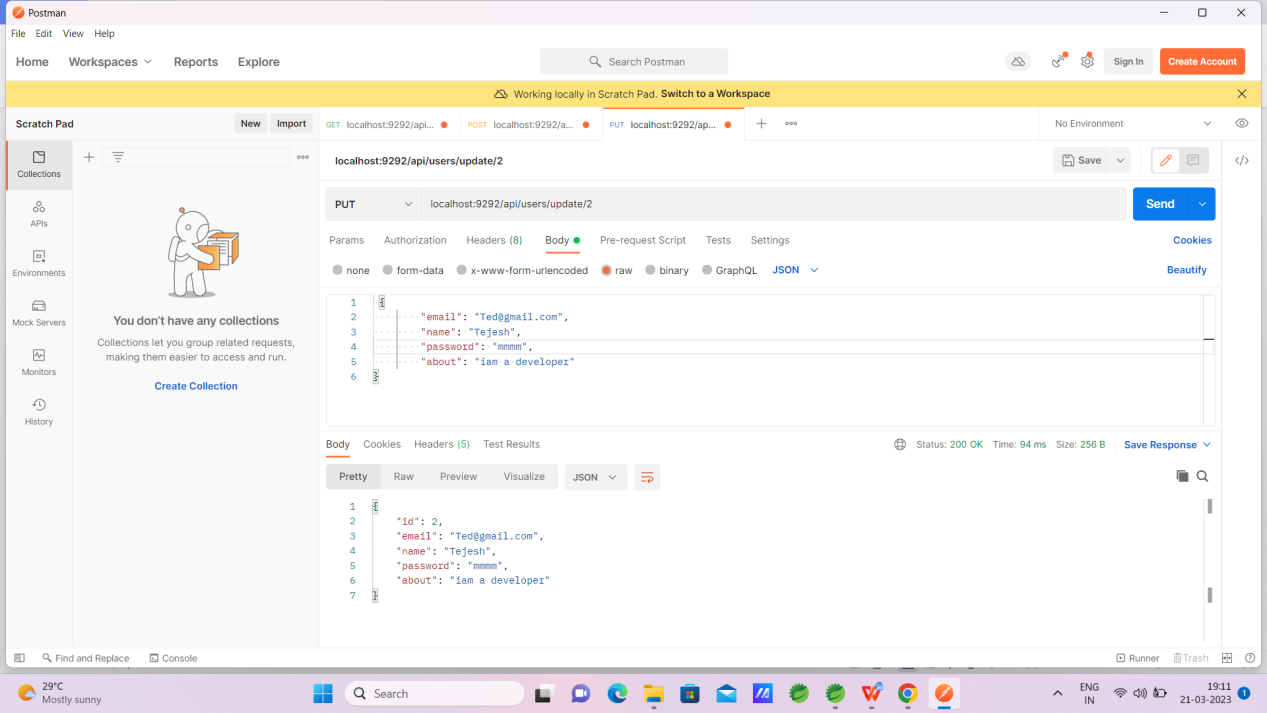
**Fetch all data**



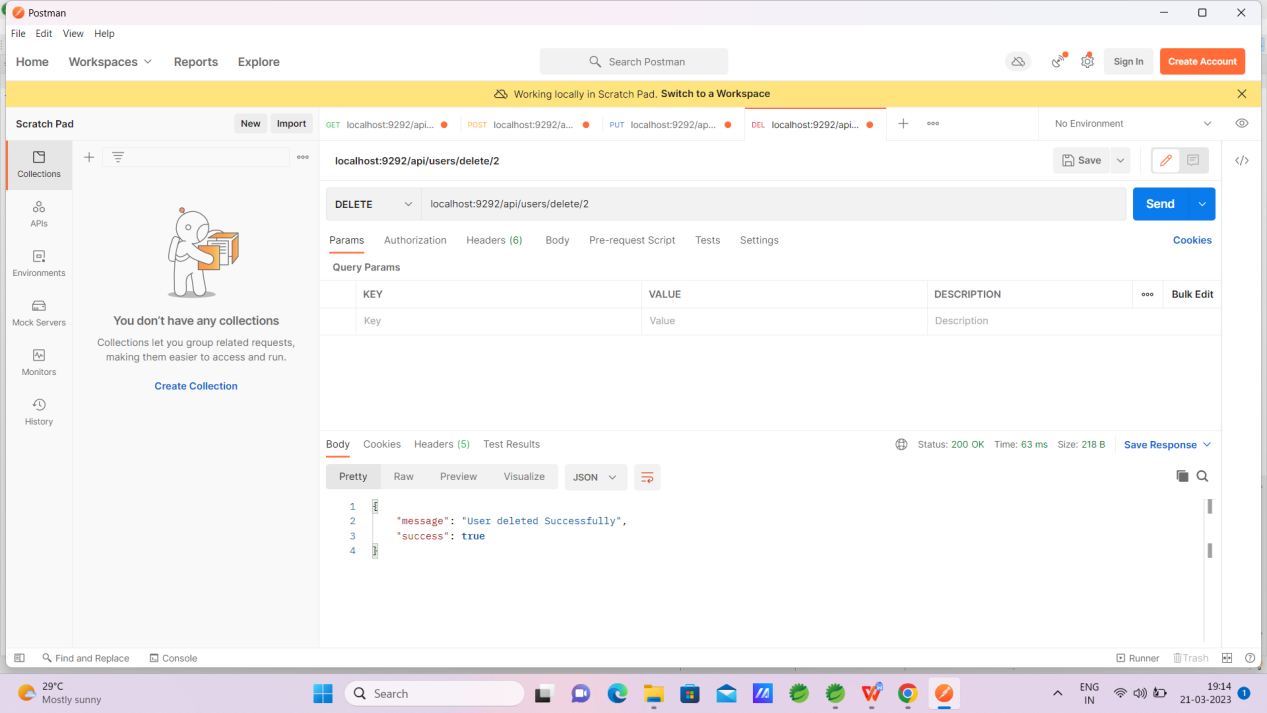
**Insert the data**



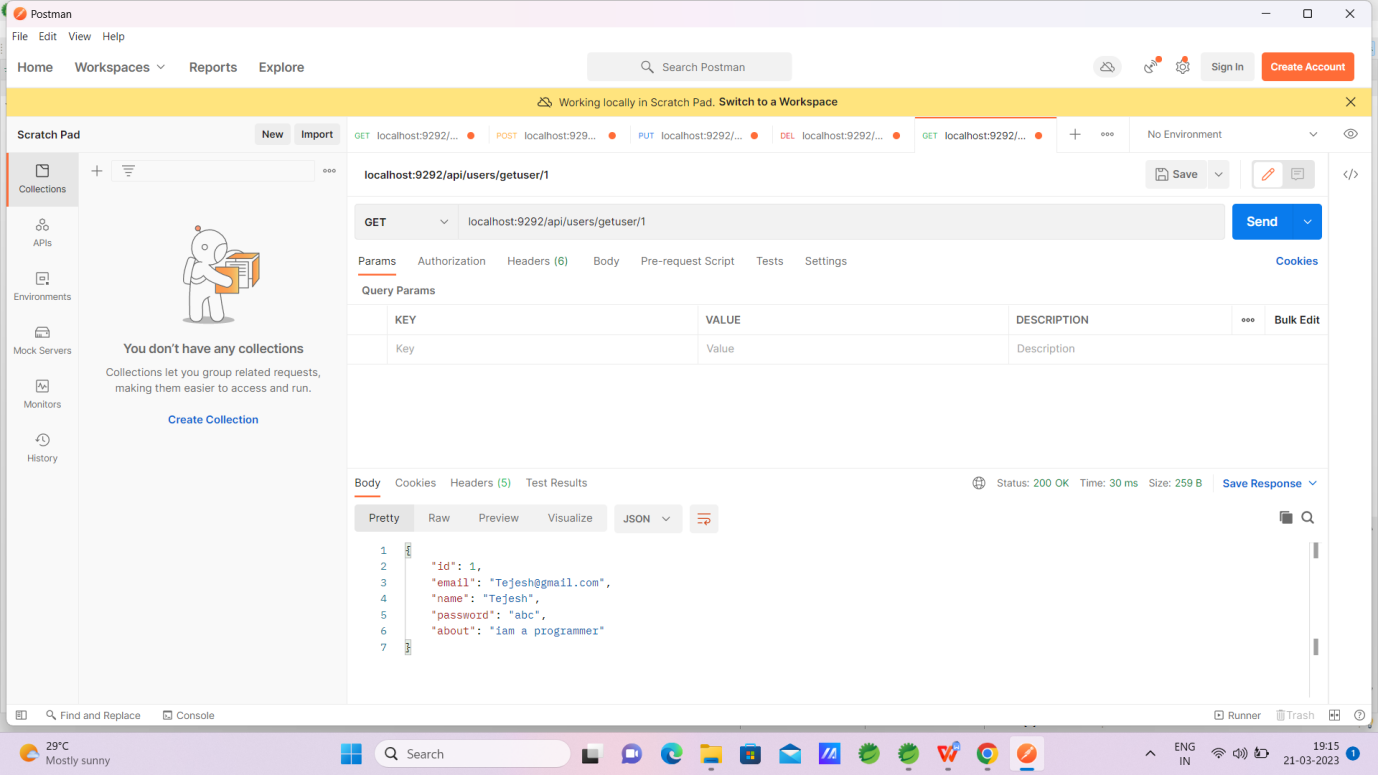
**Update the data**



**Delete the data**



**Fetch single record**



ER diagram

